The Consultative Committee for Space Data Systems

Draft Recommendation for
Space Data System Standards

THE DATA DESCRIPTION LANGUAGE EAST SPECIFICATION (CCSD0010)

DRAFT RECOMMENDED STANDARD

CCSDS 644.0-P-2.1

PINK SHEETS

February 2007

3.2 LOGICAL DESCRIPTION

The logical part of an EAST DDR is composed of:

- the logical description of the models of data (using type and subtype declarations for the syntactic definition of the data, and using representation clauses for the specification of their size in bits and their location within the set of data);
- the declaration of the data occurrences, i.e., the declaration of the described data items (using object declarations).

The logical part of the Data Description Record consists of a package. This unit is introduced by the keyword **package**, followed by the package name, and ends with '**end** package name**;**'. The package name is an identifier (see 3.1.3).

The logical description package identification must be followed by the mention of the version of the EAST recommendation to which the description is supposed to conform.

As the notion of EAST recommendation version was not present in the first two EAST recommendation issues, the absence of the mention in a description should be interpreted as a reference to these two first versions (fully compatible).

A description that conforms to a particular version of the EAST recommendation must remain correct with regard to the following versions of EAST.

If an EAST description is generated using a tool, it is recommended that the tool indicate its own version using a comment.

Types are models, and objects are instances (or occurrences) of these models. Type declarations describe therefore the structure of the data elements which may occur in the described data, while **the actual data occurrences are represented by the declaration of variables and constants**.

A type (except predefined type), a subtype or a constant (except predefined constant) must be declared in the package before being used.

The declaration of variables must occur in the latter section of the logical description. Constants may be declared in the type declaration section or in the section for the declaration of variables: in the first section, they contribute to data models definition, while they represent data occurrences in the second section.

The described data is a concatenation of elements in the order of the corresponding variables. The types used in the declaration of variables must have been previously declared in the package.

Figure 3-13 summarizes the content of the logical part of a DDR.

package *logical_package_name* **is**

EAST version and -- tool version (optional)

Section for the Declaration of Types: Definition of the Data Models

- type declarations and representation clauses (see 3.2.1 and 3.2.4)
- subtype and constant declarations (see 3.2.2 and 3.2.3.2)

Section for the Declaration of Variables: Definition of the Data Occurrences

- variable and constant declarations (see 3.2.3.1 and 3.2.3.2)
- actual values of discriminant (see 3.2.1.6)

end *logical_package_name* ;

Figure 3-13: Logical Part Structure

The version declaration should respect the following format:

east_version : constant STRING := "3.0";

-- tool version : OASIS 5.0 (optional comment)

3.2.1 TYPE DECLARATIONS

The type is characterized by a set of permissible values. Several classes of types exist: scalar types (enumeration types, integer types, and real types), array types, and record types. Some types are EAST predefined types (see 3.2.1.1); the other types are user defined types and must be declared according to a specific syntax (see 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5 and 3.2.1.6).

3.2.1.1 Predefined Types

There are three predefined types provided by the EAST language: CHARACTER, STRING and EOF. Predefined means that no previous declaration has to be made explicitly by the user to use one of these types.

The predefined type CHARACTER is an enumeration type (see next subsection for the enumeration definition syntax rules), whose values are the 256 characters of the 8-bit coded Latin Alphabet No. 1. character set (see annex B and reference [1]).

The values of the predefined type STRING are one-dimensional arrays of the predefined type CHARACTER, indexed by values in increments of one of any positive integer type.

An index is specified as follows in figure 3-19:

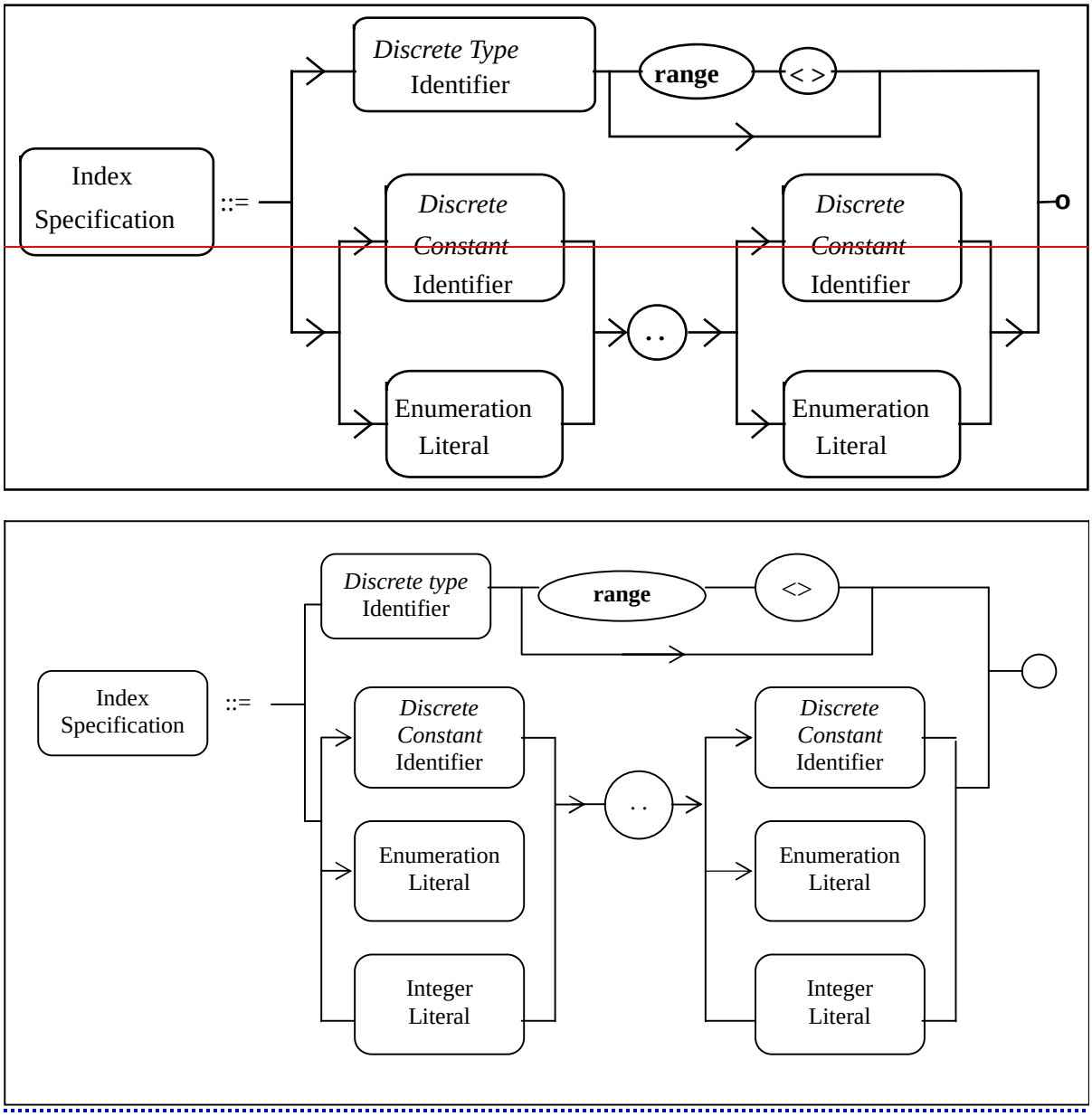


Figure 3-19: Index Specification Diagram

In the '..' notation, the first identifier or literal specifies the lower bound, while the second one specifies the upper bound.

The 'range <>' expression denotes an undetermined number of elements.

where a component declaration is specified as in figure 3-21:

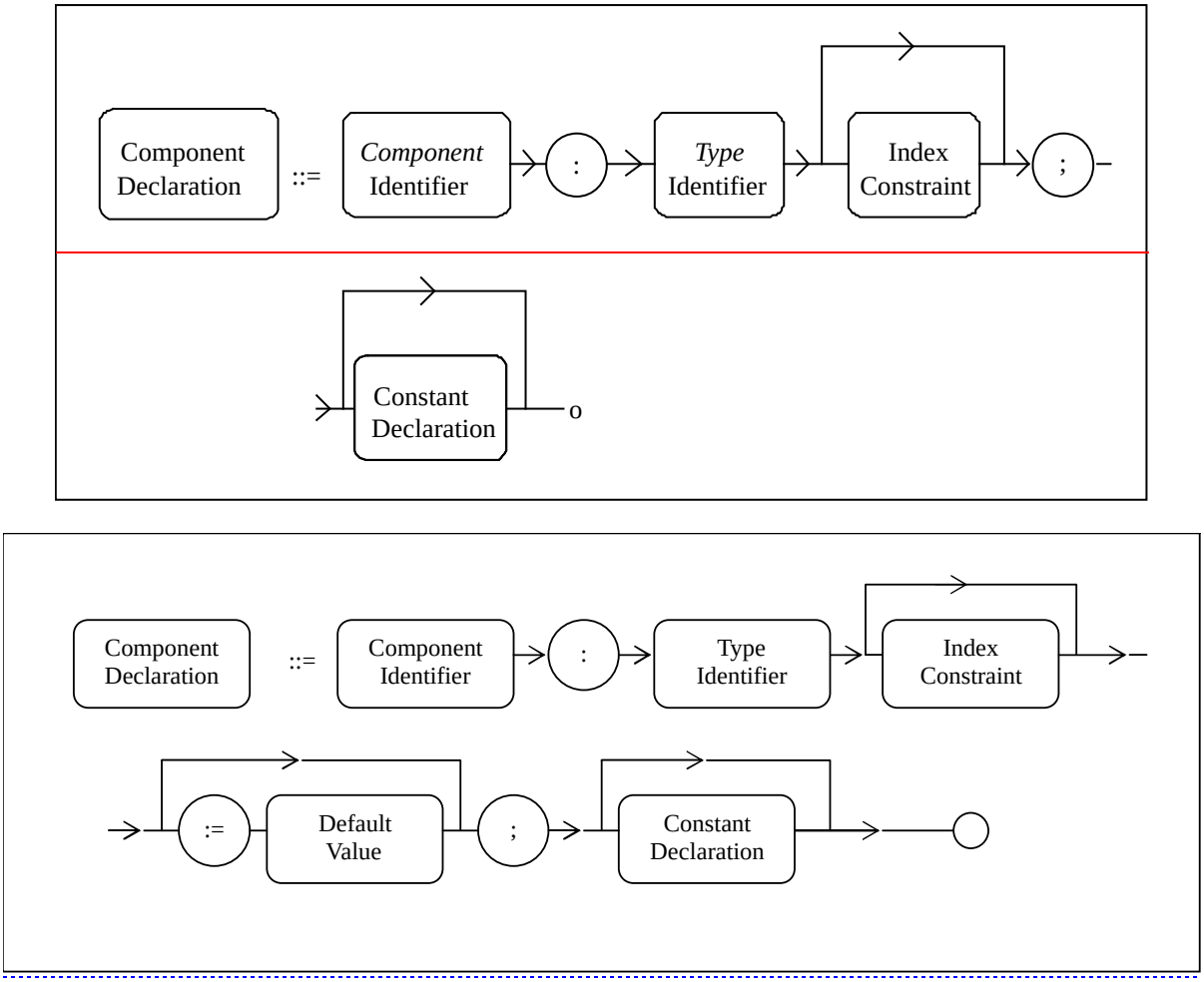


Figure 3-21: Component Declaration Diagram

The optional default value is the one to be given automatically if no other value is given by an application generating such data; it is to be used by generic software layer.

Figure 3-22 illustrates default value definitions.

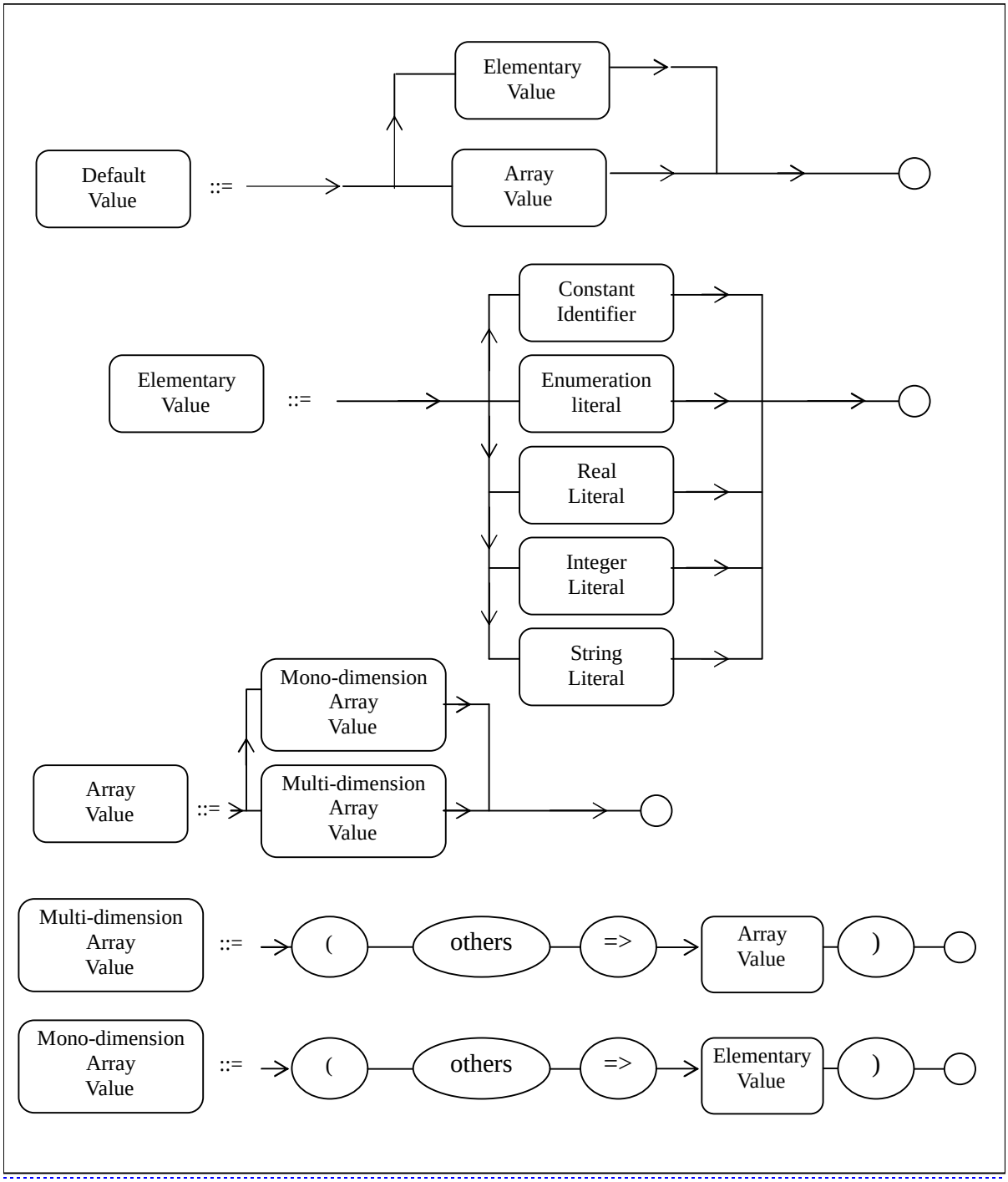


Figure 3-22: Default Value Definition Diagram

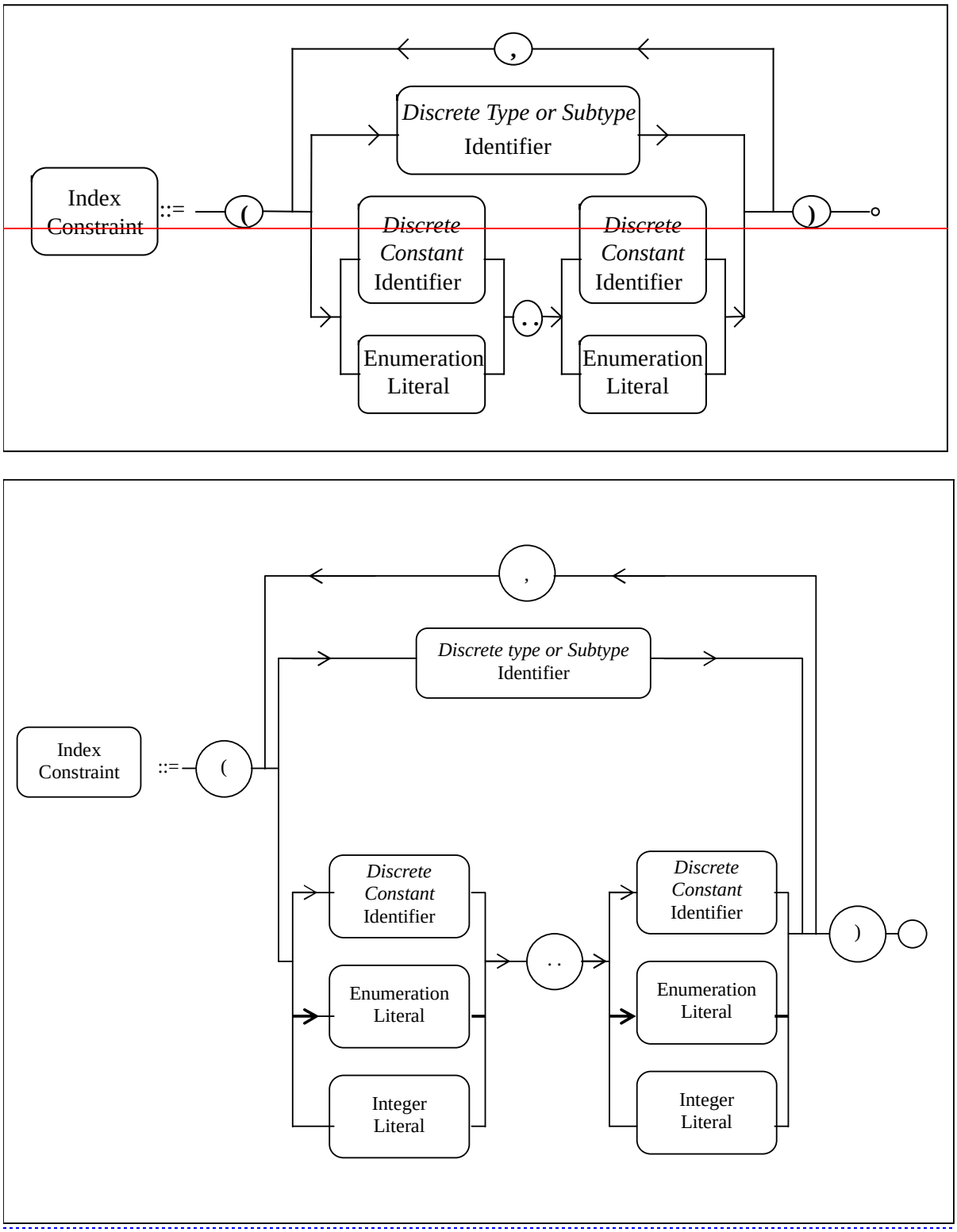


Figure 3-23: Index Constraint Diagram

The following example presents two record type definitions that consist only of simple component declarations:

```
type COMPLEX is record
        REAL_PART:  REAL;
        IMAGINARY_PART:  REAL;
end record;
-- REAL is a real type defined in 3.2.1.4 as:
-- type REAL is digits 15;
type MEASUREMENT_BLOCK is record
        TODAY:  DAY := MON;
        TEMPERATURE:  SMALL_INTEGER := 0;
        VOLUME:  SMALL_INTEGER := 0;
        FIRST_SEQUENCE_OF_MEASUREMENTS:  VECTOR(1 .. 100) := (others => 1);
        SECOND_SEQUENCE_OF_MEASUREMENTS:  VECTOR(1 ..10) := (others => 1);
end record;
-- DAY is an enumeration type defined in 3.2.1.2 as:
-- type DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);
-- SMALL_INTEGER is an integer type defined in 3.2.1.3 as:
-- type SMALL_INTEGER is range -10 .. 10;
-- VECTOR is an array type defined in 3.2.1.5 as:
-- type VECTOR is array (NUMBER range <>) of REAL;
```

Example 3-8: Record Type Definitions

Some records may contain components of which the size or even the existence depends on the value of another component, called a *discriminant*. The type of a discriminant must be discrete. Figure 3-24 illustrates the syntax of a discriminant specification.

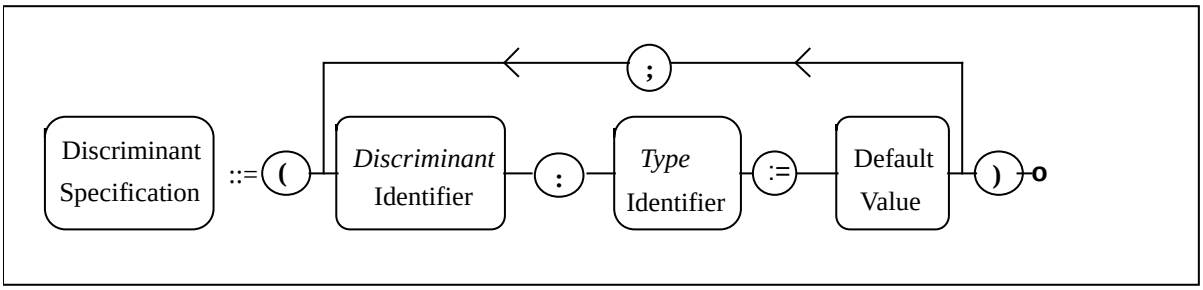


Figure 3-24: Discriminant Specification Diagram

```
                                .../...

-- structuring types
type DATA_ARRAY is array (NUMBER range <>) of OCTET;
type SECONDARY_HEADER_TYPE is array (1 .. 4) of OCTET;

type PRIMARY_HEADER_TYPE is record
  PACKET_IDENTIFICATION:  PACKET_IDENTIFICATION_TYPE;
  PACKET_SEQUENCE_CONTROL:  PACKET_SEQUENCE_CONTROL_TYPE;
  SOURCE_DATA_LENGTH:  NUMBER;
end record;

type PACKET_FORMAT_TYPE(
      VIRTUAL_SECONDARY_HEADER_FLAG:  PRESENCE_FLAG := PRESENT;
      -- point to the secondary header flag located in the first branch
      VIRTUAL_SOURCE_DATA_LENGTH:  NUMBER := 256)
      -- point to the source data length located in the third branch
is record
      PRIMARY_HEADER:  PRIMARY_HEADER_TYPE;
      case VIRTUAL_SECONDARY_HEADER_FLAG is
        when ABSENT =>
SOURCE_DATA_0:  DATA_ARRAY (1 .. VIRTUAL_SOURCE_DATA_LENGTH);
        when PRESENT =>
              SECONDARY_HEADER:  SECONDARY_HEADER_TYPE;
SOURCE_DATA_1:  DATA_ARRAY (1 .. VIRTUAL_SOURCE_DATA_LENGTH);
      end case;
end record;

FLAG : PRESENCE_FLAG;
LENGTH : NUMBER;
PACKET : PACKET_FORMAT_TYPE;
-- Actual values of discriminants
PACKET.VIRTUAL_SECONDARY_HEADER_FLAG : virtual PRESENCE_FLAG := FLAG;
PACKET.VIRTUAL_SOURCE_DATA_LENGTH : virtual NUMBER := LENGTH;
```

Example 3-11: Logical Description of the Packet Format

The two virtual discriminants 'VIRTUAL_SECONDARY_HEADER_FLAG' and 'VIRTUAL_SOURCE_DATA_LENGTH' do not really exist in the exchanged data block. They serve as a link between other data:

– VIRTUAL_SECONDARY_HEADER_FLAG is supposed to have the value of the SECONDARY_HEADER_FLAG field in the PACKET IDENTIFICATION block; it conditions the existence of the SECONDARY_HEADER block. It serves as a link between these two fields.

– VIRTUAL_SOURCE_DATA_LENGTH is supposed to have the value of the SOURCE_DATA_LENGTH field in the PRIMARY HEADER; it conditions the size of the SOURCE DATA block. It also serves as a link.

If the size of an array is deduced from several discriminants by a calculation its virtual size declaration remains unchanged (as shown on example 3-10). The calculation to be done is described later after the object declaration section (see 3.2.3) as shown in example 3-12.

```
type A_JULIAN_DAY is range 1 .. (2**32)-1;
type A_SECOND_IN_A_DAY is range 0 .. 86399;

type A_JULIAN_DATE is record
   DAY : A_JULIAN_DAY;
   SECOND : A_SECOND_IN_A_DAY;
end record;

type A_TEMPERATURE is digit 6 range 0.0 .. 100.0;

type TEMPERATURES  is array (A_JULIAN_DAY range <> ) of A_TEMPERATURE;

type DATA_RECORD (VIRTUAL_SIZE : A_JULIAN_DAY := 1) is record
      MEASUREMENTS : TEMPERATURES (1 .. VIRTUAL_SIZE);
end record;

FIRST_DATE : A_JULIAN_DATE;
LAST_DATE : A_JULIAN_DATE;
DATA : DATA_RECORD;
-- Actual values of discriminant
DATA.VIRTUAL_SIZE : virtual DAY_TYPE := LAST_DATE.DAY -
FIRST_DATE.DAY;
```

Example 3-12: Calculated Size Array

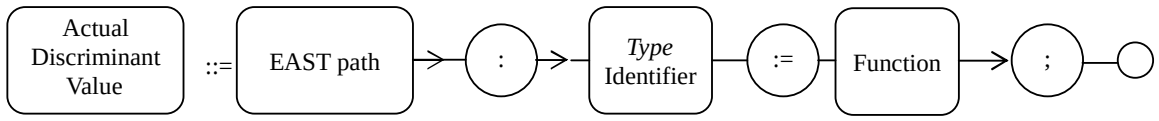
Supported operators are ‘+’, ‘-’, ‘*’, ‘/’, ‘**’ (exponent), ‘is_odd’, ‘is_even’, ‘cos’, ‘sin’, ‘tan’, ‘acos’, ‘asin’, ‘atan’, ‘log’, ‘ln’, ‘cosh’, ‘sinh’, ‘tanh’, ‘acosh’, ‘asinh’, ‘atanh’, ‘(’, ‘)’, ‘!’ (factorial).

The syntax of the virtual declaration for a calculated contition is the same (as in example 3-9).

The calculation to be done is described later after the object declaration section.

```
type A_RESULT is range 0 .. 100;

type RESULTS (VIRTUAL_BONUS_FLAG : BOOLEAN := TRUE) is record
   RESULT_1 : A_RESULT;
   RESULT_2 : A_RESULT;
   case VIRTUAL_BONUS_FLAG is
      when TRUE => BONUS : A_RESULT;
   end case;
end record;

PREVIOUS_WEEK : A_RESULT;
THIS_WEEK : RESULTS;

-- Actual values of discriminant

THIS_WEEK.VIRTUAL_BONUS_FLAG : virtual BOOLEAN
                    := (THIS_WEEK.RESULT_2 - THIS_WEEK.RESULT_1) > PREVIOUS_WEEK;
```

Example 3-13: Calculated Component Presence Condition



NOTES

1 Function returns a value compliant with type identifier computed using predefined operators applied to values designated by their complete EAST path.

2 The EAST path is built using the name of each hierarchy level from the top to the designated item.

Figure 3-27: Actual Discriminant Value Declaration Diagram

3.2.3 OBJECT DECLARATIONS

An object is an entity that contains a value of a given type. A declared object is called a constant if the reserved word **constant** appears in the object declaration. An object that is not a constant is called a variable.

3.2.3.1 Declaration of Variables

The declaration of a variable uses the previous type, subtype, or constant declarations. Variables correspond to the data that are to be exchanged. Figure 3-33 illustrates the syntax for the declaration of a variable.

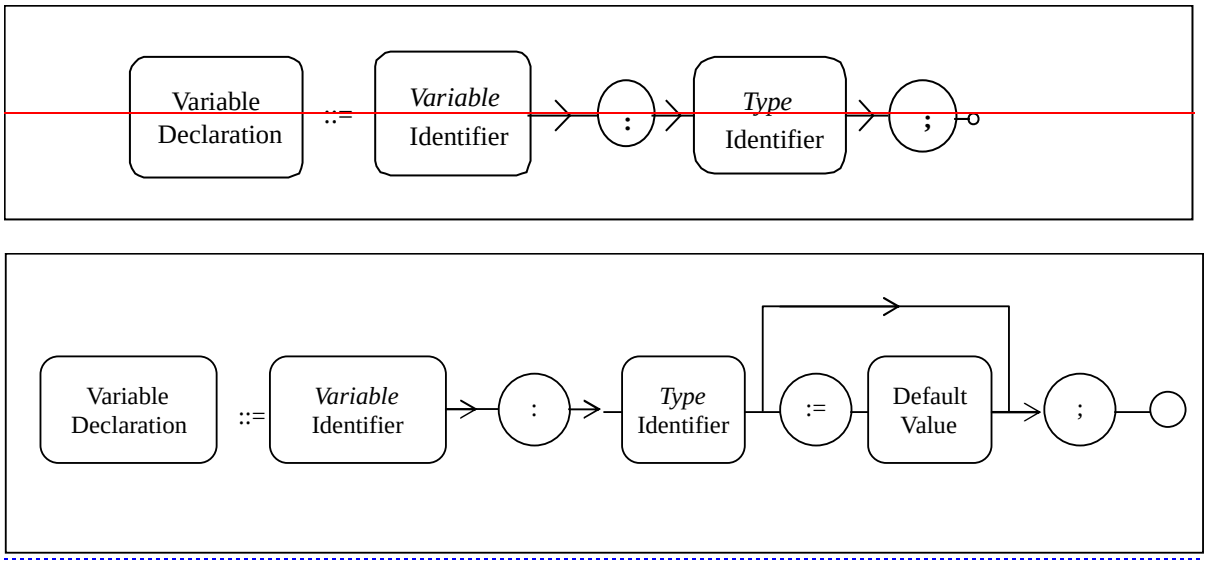


Figure 3-33: Variable Declaration Diagram

The default value (which definition is given by figure 3-22) is the one to be given automatically if no other value is given by an application generating such data; it is to be used by generic software layer.

A variable declaration consists of only one identifier (the variable identifier) followed by the identifier of the type that describes the corresponding data.

```
    UPDATED_DATA:  MEASUREMENT_BLOCK ;
       -- MEASUREMENT_BLOCK is a record type defined in 3.2.1.6
    INSTRUMENT_STATUS : STATE := ON;
       -- STATE is an enumeration type defined in 3.2.1.2:
       -- ON is a default value
```

Example 3-16: Variable Declaration

The following example (figure 3-40) presents the case of an incomplete record representation clause. A fortiori no representation clause could be found after a computed size array or a computed structure record.

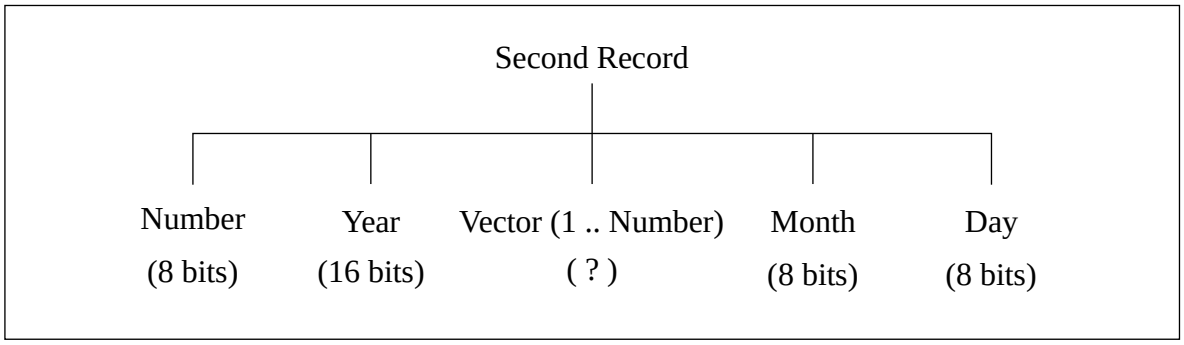


Figure 3-40: Second Tree Structure

The number of measurements is not known at definition time. The size of the vector of measurements is therefore not provided. The tree structure is described using the following declarations:

```
type SECOND_RECORD(THE_NUMBER:  NUMBER := 1) is record
        THE_YEAR:  YEAR;
        THE_MEASUREMENT:  VECTOR(1 .. THE_NUMBER);
        THE_MONTH:  MONTH;
        THE_DAY_OF_MONTH:  DAY;
end record;
for SECOND_RECORD use record
        THE_NUMBER at 0 range 0 .. 7;
        THE_YEAR at 0 range 8 .. 23;
        -- no component clause for THE_MEASUREMENT,
        -- for THE_MONTH nor for THE_DAY_OF_MONTH
end record;
-- no length clause for SECOND_RECORD type
```

Example 3-27: Incomplete Record Representation Clause Declaration

In this example, the length of ‘THE_MEASUREMENT’ depends on the value of the discriminant ‘THE_NUMBER’. No representation clause can be given for it. Nevertheless the size is determined by the expression ‘THE_NUMBER times 32’, 32 being the size of the basic element VALUE. The component ‘THE_MEASUREMENT’ begins at bit 24. The length of ‘THE_MONTH’ is known but its location is not known at definition time. No representation clause can be given for it. The component ‘THE_MONTH’ begins after the end of ‘THE_MEASUREMENT’. In the same way, the length of ‘the_day_of_month’ is known, but its location is not known at definition time. No representation clause can be given for it. The component ‘THE_DAY_OF_MONTH’ begins after the end of ‘THE_MONTH’.

The actual representation of the numerics is given by the declaration of constants of the previous record types (INTEGER_PHYSICAL_DESCRIPTION for the representation of integers and REAL_PHYSICAL_DESCRIPTION for the representation of reals).

The actual representation of a numeric is therefore provided by a record value (i.e., the value of the constant of the relevant record type: INTEGER_PHYSICAL_DESCRIPTION or REAL_PHYSICAL_DESCRIPTION).

Figure 3-44 illustrates the syntax of a record value.

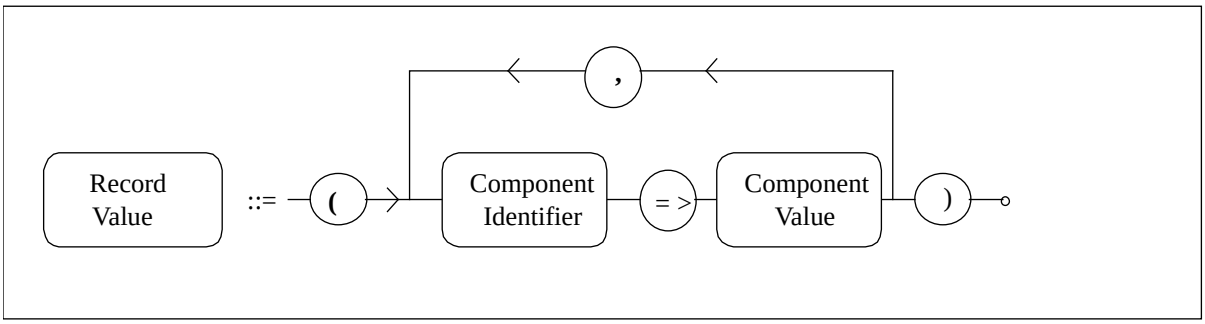


Figure 3-44: Record Value Specification Diagram

In the case of the record types used in the physical part of an EAST description, the component value is either an enumeration literal, an integer literal or an array value (see figure 3-45).

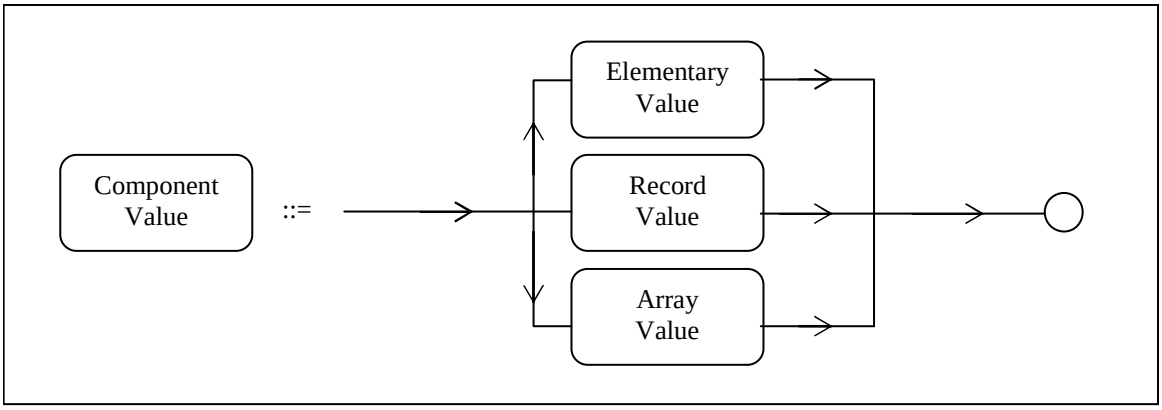


Figure 3-45: Component Value Definition Diagram

4 RESERVED KEYWORDS

The following reserved keywords are not available for use as declared identifiers. Some of them are reserved keywords of the Ada programming language (see reference [E3]), but not of the EAST language. These words are also reserved in order to avoid any problem in the case of an Ada application accessing the data. Other words are reserved identifiers of the EAST language and not of the Ada programming language.

a) EAST and Ada Keywords

array	digits	is	package	type
at				
	end	null	range	use
case			record	
constant	for	of		when
		others	subtype	

b) Other Ada Keywords

abort	delta	if	pragma	tagged
abs	do	in	private	task
abstract			procedure	terminate
accept	else	limited	protected	then
access	elsif	loop		
aliased	entry		raise	until
all	exception	mod	rem	
and	exit		renames	while
		new	requeue	with
begin	function	not	return	
body			reverse	xor
	generic	or		
declare	goto	out	select	
delay			separate	

c) Pure EAST reserved identifiers

virtual_...	word_32_bits	word_16_bits	east_version	virtual

NOTE – Any identifier beginning with ‘virtual_’ is reserved for virtual component identifier only.

In Ada, a pragma is used to convey information to Ada compilers. As such, pragma use is not justified in EAST.

D2 ADA SYNTAX ELEMENTS THAT HAVE A DIFFERENT MEANING IN EAST

A length clause is defined by the following declaration:

for type_identifier'**size use** static_expression **;**

In Ada, the value of the expression specifies an upper bound for the number of bits to be allocated to objects of the given type. In EAST, the expression specifies the exact number of bits that any object of the given type occupies.

In Ada, a record representation clause specifies the storage representation of records in memory, that is, the order, position, and size of record components in memory of a given machine. In EAST, the record representation clause specifies the actual storage representation on the medium.

D3 EAST SYNTAX RESTRICTIONS VS. ADA

In Ada, the base for based numeric literals can be any number between 2 and 16. In EAST the base can only be 2, 8 or 16.

In Ada the values specified in a range constraint within an integer or real type definition can be a simple_expression. ~~In EAST the values may only be a numeric literal or an identifier naming an appropriate numeric constant.~~In EAST the values may only be a numeric literal or an identifier (naming an appropriate numeric constant or an appropriate discriminant eventually computed later, as described in 3.2.1.6).

In Ada, a constant declaration allows a list of identifiers. EAST allows only a single identifier.